Matt Matuk

CSIT – 211

14 March 2015

Lab 6

**Programming Project PP 15.15**

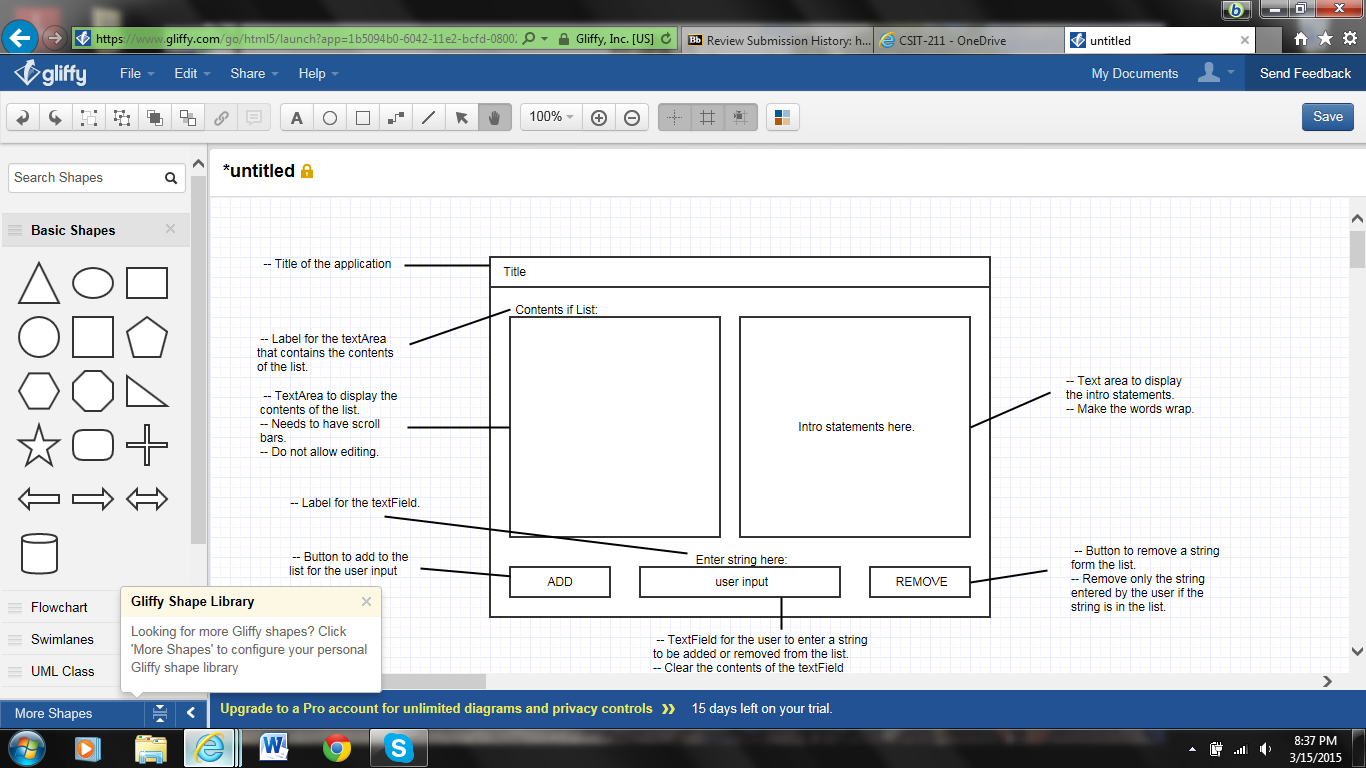
-----**SPEC**-----

Create a graphical application that provides a button for add and remove from an ordered list, a text field to accept a string as input for add, and a text area to display the contents of the list after each operation.

**SCRUM**

* Class FrameDoubleLinkedList
  + Methods
    - Main()
* Class PanelDoubleLinkedList
  + Variables
    - DoubleOrderList<String> list
  + Methods
    - Constructor()
  + Class ButtonListener implements ActionListener
    - Methods
      * ActionEvent(Event e)
* Class DoubleOrderList extends DoubleList
  + Variables
    - Private Int count
    - Private DoubleNode<T> head
    - Priavte DoubleNode<T> tail
    - Private final static int NOT\_FOUND = -1
    - Private int modCount
  + Method
    - DoubleOrderedList()
    - Add(T element)
    - Private int find(T target)
    - Bool contains(T target)
* Class DoubleNode<T>
  + Variables
    - Private DoubleNode<T> next
    - Private DoubleNode<T> previous
    - Private T element
  + Methods
    - DoubleNode()
    - DoubleNode(T elem)
    - setNext(DoubleNode<T> node)
    - setPrevious(DoubleNode<T> node)
    - DoubleNode<T> getNext()
    - DoubleNode<T> getPrevious()
    - T getElement()
    - setElement(T elem)
* Class DoubleList
  + Methods
    - T removeFirst()
    - T removeLast()
    - T remove(T element)
    - T first()
    - T last()
    - Bool isEmpty()
    - Int size()
    - Iterator iterator()
    - String toString()
  + Class Private DoubleIterator
    - Variable
      * Private int iteratorModCount
      * Private DoubleNode<T> current
    - Methods
      * DoubleIterator()
      * Bool hasNext()
      * T next()
      * Remove()

**GUI Storyboard**



**Class: FrameDoubleLinkedList**

1. This class will be the driver that creates the frame which will hold the panels

**Import**

**Process**

**Class: PanelDoubleLinkedList**

1. This class will create the panel that hold the text for intro, the text area that contains strings in the list, the two buttons to add and remove from the list, a text field for the user to enter strings, and any labels need to make the application more readable.
2. Keep the lyout similar to the storyboard, but the color and size can be change if needed.
3. All methods are public and void unless otherwise noted
4. All variables are protected unless otherwise noted.

**Import**

* All swing and awt
* Queue

**Process**

* Variables
  + Two buttons
    - One for add
    - One for remove
  + Two text areas
    - One for the contents of the list
    - One for the intro text for the application
  + Text field
    - User input
  + Any labels needed
  + New doubleOrderedList
    - Contains the strings entered by the user
* Methods
  + Constructor
    - Create panels to hold the info
    - Need to layout the application to look like the storyboard
    - Create a button listener class and add the two buttons to the listener
    - When the add button is pressed
      * Add the user entered string to the list and update the text area to display the new contents of the list
    - When the remove button is pressed
      * Check to see if the string entered is in the list
        + If not, display a message saying the string entered is not in the list
        + If in the list, remove the string and update the text area to display the new contents of the list

**Class: DoubleOrderList extends DoubleList**

1. This class extends the DoubleList class and adds the add method to the class.
2. The add method will add items to the list in the proper order
3. All methods and variables are generic
4. All methods are public unless otherwise stated

**Import**

**Process**

* Method
  + DoubleOrderedList()
    - Count = 0
    - Head = tail = null
  + Add(T element)
    - If the element is not comparable
      * Throw NonComparableElementExcepetion(“OrderedList”)
    - Create a new DoubleNode newNode and set the element to the passed element
    - Create a new Comparable object comparableElement
    - Create a new DoubleNode currentNode = head
    - Int scan = 0
    - Find the insertion point by looping while scan is less than count and while the comparableElment when comparaed to the current nodes element is greater than 0
      * currentNode = currentNode’s next Node
    - Create a new DoubleNode previous set equal to currentNode getPrevious()
    - newNode setPrevious() to previous
    - newNode setNext() to currentNode
    - previous setNext() to newNode()
    - currentNode setPrevious() to newNode()
    - increase mod count

**Class: DoubleNode<T>**

1. This class is a double node that will contains a reference to the next and previous nodes.
2. This class will also contain a element
3. All methods and variables are generic

**Import**

**Process**

* Variables
  + Protected DoubleNode<T> next
  + protected DoubleNode<T> previous
  + protected T element
* Methods
  + DoubleNode()
    - Element = null
    - Next = previous = null
  + DoubleNode(T elem)
    - Element = elem
    - Next = previous = null
  + setNext(DoubleNode<T> node)
    - next = node
  + setPrevious(DoubleNode<T> node)
    - previous = node
  + DoubleNode<T> getNext()
    - return next
  + DoubleNode<T> getPrevious()
    - Return previous
  + T getElement()
    - Return element
  + setElement(T elem)
    - element = elem

**Class: DoubleList<T>**

1. This class will contain all the methods needed to create a double linked list
2. The class will be the parent class for DoubleOrderList
3. All methods are public and void unless otherwise stated
4. All methods and variables are generic unless otherwise stated

**Import**

**Process**

* Variables
  + Private Int count
  + Private DoubleNode<T> head
  + Priavte DoubleNode<T> tail
  + Private int modCount
* Methods
  + T removeFirst()
    - Need to return the head element
    - Set head equal to head.getNext()
  + T removeLast()
    - Need to return the tail element
    - Set the tail equal to tail.getPrevious()
  + T remove(T element)
    - Need to return the element removed
    - Use the find() method to check to see if the element is in the list
    - Throw exception if not found
    - If the element is found
      * Rest the nodes before and after to point to the new nodes
      * Decrease count by 1
      * Increase modCount by 1
  + T first()
    - Return head
  + T last()
    - Return tail
  + Bool isEmpty()
    - Return true if count = 0
  + Int size()
    - Return count
  + Iterator iterator()
  + Private DoubleNode<T> find(T target)
    - Need int scan = 0 and DoubleNode<T> result = null
    - If the list is not empty
      * Loop while result is equal to null and scan is less than the size of the list
        + If the target equals the element

Result = element

* + - * + Else

Scan increase by one

* + - Return result
  + Bool contains(T target)
    - Return the result of (find(target) != null)
  + String toString()
* Class Private DoubleIterator
  + Variable
    - Private int iteratorModCount
    - Private DoubleNode<T> current
  + Methods
    - DoubleIterator()
      * Current = head
      * iteratorModCount = modCount
    - Bool hasNext()
      * If the mod counts are different
        + Throw exception
      * Return (current != null)
    - T next()
      * If hasNext is false
        + Throw excepetion
      * T result = current’s element
      * Set current equal to current’s next node
      * Return result
    - Remove()
      * Not supported

**CODE**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// FrameDoubleLinkedList.java Matt Matuk

// CSIT 211 Project 15.15 pg 616

// 1. This class creates a new PanelDoubleOderList and frame

// for the application

// 2. This class is the main driver class

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**import** java.awt.Dimension;

**import** javax.swing.JFrame;

**public** **class** FrameDoubleLinkedList

{

**public** **static** **void** main(String[] args)

{

JFrame frmStack = **new** JFrame("Double Linked Ordered Stack");

frmStack.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

frmStack.setMinimumSize(**new** Dimension(450,300));

frmStack.setBounds(100, 100, 450, 300);

frmStack.getContentPane().add(**new** PanelDoubleLinkedList());

frmStack.pack();

frmStack.setVisible(**true**);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// PanelDoubleLinkedList.java Matt Matuk

// CSIT 211 Project 15.15 pg 616

// 1. This class will contain the method that will create all

// the components of the panel for the frame

// 2. Make sure when creating the window, that it looks similar

// to the GUI storyboard. Color and exact size can change.

// 3. Multiple methods will be used to add and remove strings

// from the double linked ordered list

// 4. All methods are public and void unless otherwise stated

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**import** javax.swing.\*;

**import** javax.swing.border.TitledBorder;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.util.NoSuchElementException;

@SuppressWarnings("serial")

**public** **class** PanelDoubleLinkedList **extends** JPanel

{

**private** JTextField txtFieldUserInput;

**protected** JButton btnRemove, btnAdd;

**protected** DoubleOrderList<String> list = **new** DoubleOrderList<String>();

**protected** JTextArea txtAreaList;

**protected** JPanel panel;

//-------------------------------------------------------------------

// This method create all the panels and layouts for the application.

// This method also sets any listeners and creates the view.

//-------------------------------------------------------------------

**public** PanelDoubleLinkedList()

{

setBorder(**new** TitledBorder(**null**, "List", TitledBorder.***LEADING***, TitledBorder.***TOP***, **null**, **null**));

SpringLayout springLayout = **new** SpringLayout();

setLayout(springLayout);

panel = **new** JPanel();

springLayout.putConstraint(SpringLayout.***NORTH***, panel, 0, SpringLayout.***NORTH***, **this**);

springLayout.putConstraint(SpringLayout.***WEST***, panel, 0, SpringLayout.***WEST***, **this**);

springLayout.putConstraint(SpringLayout.***SOUTH***, panel, -30, SpringLayout.***SOUTH***, **this**);

springLayout.putConstraint(SpringLayout.***EAST***, panel, 0, SpringLayout.***EAST***, **this**);

add(panel);

JPanel panel\_1 = **new** JPanel();

springLayout.putConstraint(SpringLayout.***NORTH***, panel\_1, 0, SpringLayout.***SOUTH***, panel);

SpringLayout sl\_panel = **new** SpringLayout();

panel.setLayout(sl\_panel);

JPanel panel\_2 = **new** JPanel();

sl\_panel.putConstraint(SpringLayout.***NORTH***, panel\_2, 10, SpringLayout.***NORTH***, panel);

sl\_panel.putConstraint(SpringLayout.***WEST***, panel\_2, 10, SpringLayout.***WEST***, panel);

sl\_panel.putConstraint(SpringLayout.***SOUTH***, panel\_2, -22, SpringLayout.***SOUTH***, panel);

panel.add(panel\_2);

JLabel lblUserString = **new** JLabel("Please enter your string here.");

sl\_panel.putConstraint(SpringLayout.***NORTH***, lblUserString, 6, SpringLayout.***SOUTH***, panel\_2);

sl\_panel.putConstraint(SpringLayout.***WEST***, lblUserString, 106, SpringLayout.***WEST***, panel);

sl\_panel.putConstraint(SpringLayout.***EAST***, lblUserString, -95, SpringLayout.***EAST***, panel);

panel.add(lblUserString);

JTextArea txtAreaIntro = **new** JTextArea();

sl\_panel.putConstraint(SpringLayout.***NORTH***, txtAreaIntro, 10, SpringLayout.***NORTH***, panel);

sl\_panel.putConstraint(SpringLayout.***WEST***, txtAreaIntro, 225, SpringLayout.***WEST***, panel);

sl\_panel.putConstraint(SpringLayout.***SOUTH***, txtAreaIntro, -6, SpringLayout.***NORTH***, lblUserString);

sl\_panel.putConstraint(SpringLayout.***EAST***, panel\_2, -6, SpringLayout.***WEST***, txtAreaIntro);

txtAreaIntro.setLineWrap(**true**);

txtAreaIntro.setBackground(SystemColor.***menu***);

sl\_panel.putConstraint(SpringLayout.***EAST***, txtAreaIntro, -10, SpringLayout.***EAST***, panel);

txtAreaIntro.setWrapStyleWord(**true**);

txtAreaIntro.setEditable(**false**);

txtAreaIntro.setText("Welcome to my application. Today you will be "

+ "entering a string at that bottom of the page and then "

+ "press ADD or REMOVE to either add or remove the string from "

+ "your ordered list. The contents of your ordered list will be "

+ "displayed to the left.");

panel.add(txtAreaIntro);

panel\_2.setLayout(**new** BoxLayout(panel\_2, BoxLayout.***Y\_AXIS***));

JLabel lblList = **new** JLabel("Here is the contents of your list: ");

lblList.setFont(**new** Font("Tahoma", Font.***PLAIN***, 13));

lblList.setAlignmentX(Component.***CENTER\_ALIGNMENT***);

panel\_2.add(lblList);

sl\_panel.putConstraint(SpringLayout.***NORTH***, lblList, 0, SpringLayout.***NORTH***, panel\_2);

sl\_panel.putConstraint(SpringLayout.***WEST***, lblList, 30, SpringLayout.***EAST***, panel\_2);

Component verticalStrut = Box.*createVerticalStrut*(5);

panel\_2.add(verticalStrut);

txtAreaList = **new** JTextArea();

JScrollPane scrollPane = **new** JScrollPane(txtAreaList);

panel\_2.add(scrollPane);

sl\_panel.putConstraint(SpringLayout.***WEST***, txtAreaList, 10, SpringLayout.***WEST***, panel);

sl\_panel.putConstraint(SpringLayout.***SOUTH***, txtAreaList, -6, SpringLayout.***NORTH***, lblUserString);

txtAreaList.setWrapStyleWord(**true**);

txtAreaList.setEditable(**false**);

sl\_panel.putConstraint(SpringLayout.***EAST***, txtAreaList, -6, SpringLayout.***WEST***, txtAreaIntro);

springLayout.putConstraint(SpringLayout.***WEST***, panel\_1, 0, SpringLayout.***WEST***, **this**);

springLayout.putConstraint(SpringLayout.***SOUTH***, panel\_1, -10, SpringLayout.***SOUTH***, **this**);

springLayout.putConstraint(SpringLayout.***EAST***, panel\_1, 0, SpringLayout.***EAST***, **this**);

add(panel\_1);

panel\_1.setLayout(**new** BoxLayout(panel\_1, BoxLayout.***X\_AXIS***));

Component horizontalStrut = Box.*createHorizontalStrut*(20);

panel\_1.add(horizontalStrut);

btnAdd = **new** JButton("ADD");

panel\_1.add(btnAdd);

Component horizontalStrut\_1 = Box.*createHorizontalStrut*(20);

panel\_1.add(horizontalStrut\_1);

txtFieldUserInput = **new** JTextField();

panel\_1.add(txtFieldUserInput);

txtFieldUserInput.setColumns(10);

Component horizontalStrut\_2 = Box.*createHorizontalStrut*(20);

panel\_1.add(horizontalStrut\_2);

btnRemove = **new** JButton("REMOVE");

panel\_1.add(btnRemove);

Component horizontalStrut\_3 = Box.*createHorizontalStrut*(20);

panel\_1.add(horizontalStrut\_3);

ButtonListener listener = **new** ButtonListener();

btnRemove.addActionListener(listener);

btnAdd.addActionListener(listener);

}

**private** **class** ButtonListener **implements** ActionListener

{

//-------------------------------------------------------------------

// This method handles the button clicks

//-------------------------------------------------------------------

**public** **void** actionPerformed(ActionEvent event)

{

**if** (event.getSource() == btnRemove)

{

btnRemove();

}

**else**

{

btnAdd();

}

}

//-------------------------------------------------------------------

// When the user pushes the PUSH button, the text entered in the

// text field is placed in the list

//-------------------------------------------------------------------

**private** **void** btnAdd()

{

**if** (!txtFieldUserInput.getText().equalsIgnoreCase(""))

{

**try**

{

list.add(txtFieldUserInput.getText());

}

**catch** (Exception e)

{

e.printStackTrace();

}

txtFieldUserInput.setText("");

txtAreaList.setText(list.toString());

}

}

//-------------------------------------------------------------------

// When the user presses the POP button, the the top string in the

// stack is removed and then the top line in the txtAreaStack is removed

//-------------------------------------------------------------------

**private** **void** btnRemove()

{

**if** (list.isEmpty() != **true**)

{

**try**

{

list.remove(txtFieldUserInput.getText());

txtAreaList.setText(list.toString());

}

**catch** (NoSuchElementException e)

{

JOptionPane.*showMessageDialog*(panel, "The string entered is not in the list. "

+ "Please enter only strings that are in the list to remove.");

}

}

// sets Text area to empty if stack is emtpy

**if** (list.isEmpty())

{

txtAreaList.setText("EMPTY:");

}

txtFieldUserInput.setText("");

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DoubleOrderList.java Matt Matuk

// CSIT 211 Project 15.15 pg 616

// 1. This class extends the DoubleList class and adds the add

// method to the class.

// 2. The add method will add items to the list in the proper order

// 3. All methods and variables are generic

// 4. All methods are public unless otherwise stated

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **class** DoubleOrderList<T> **extends** DoubleList<T>

{

**public** DoubleOrderList()

{

count = 0;

head = tail = **null**;

}

/\*\*

\* Adds a element to the list and places the element in the proper

\* order. Keeps the list ordered.

\*

\* **@param** element The element to be added

\* **@throws** Exception Thrown of the element is not comparable

\*/

**public** **void** add(T element) **throws** Exception

{

**if** (!(element **instanceof** Comparable))

{

**throw** **new** Exception("Element is not comparable");

}

// this is for if the list is empty

**if** (isEmpty())

{

head = **new** DoubleNode<>(element);

tail = head;

}

**else**

{

@SuppressWarnings("unchecked")

Comparable<T> comparableElement = (Comparable<T>) element;

DoubleNode<T> previous = **null**;

DoubleNode<T> current = head;

DoubleNode<T> newNode = **new** DoubleNode<>(element);

**int** scan = 0;

// loops until the element goes before the current element

// being checked

**while** ((scan < count && current != **null**) && (element **instanceof** String

? (((String)element).compareToIgnoreCase((String)current.getElement()) > 0)

: comparableElement.compareTo(current.getElement()) > 0))

{

previous = current;

current = current.getNext();

}

**if** (current == head) // if the element goes at the front

{

head.setPrevious(newNode);

newNode.setNext(head);

head = newNode;

}

**else** **if** (current == **null**) // if the element goes at the end

{

tail.setNext(newNode);

newNode.setPrevious(tail);

tail =newNode;

}

**else**

{

previous.setNext(newNode);

newNode.setPrevious(previous);

newNode.setNext(current);

current.setPrevious(newNode);

}

}

count++;

modCount++;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DoubleList.java Matt Matuk

// CSIT 211 Project 15.15 pg 616

// 1. This class will contain all the methods needed to create a

// double linked list

// 2. The class will be the parent class for DoubleOrderList

// 3. All methods are public and void unless otherwise stated

// 4. All methods and variables are generic unless otherwise stated

// 5. This class will also contain a private class for the Iterator

// 6. This class will be adstract

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**import** java.util.ConcurrentModificationException;

**import** java.util.Iterator;

**import** java.util.NoSuchElementException;

**public** **abstract** **class** DoubleList<T> **implements** Iterable<T>

{

**protected** **int** count;

**protected** DoubleNode<T> head, tail;

**protected** **int** modCount;

/\*\*

\* Removes the first element

\*

\* **@return** The element that will be removed

\*/

**public** T removeFirst()

{

**if** (!isEmpty())

{

T result = head.getElement();

DoubleNode<T> tmp = head.getNext();

tmp.setPrevious(**null**);

head = tmp;

count--;

modCount++;

**return** result;

}

**else**

{

**return** **null**;

}

}

/\*\*

\* Removes the last element

\*

\* **@return** The element removed

\*/

**public** T removeLast()

{

**if** (!isEmpty())

{

T result = tail.getElement();

DoubleNode<T> tmp = tail.getPrevious();

tmp.setNext(**null**);

tail = tmp;

count--;

modCount++;

**return** result;

}

**else**

{

**return** **null**;

}

}

/\*\*

\* Removes a certain element

\*

\* **@exception** Throws NoSuchElementException exception if the element was not found

\* **@param** element The element to be removed

\* **@return** The element that was removed

\*/

**public** T remove(T element)

{

DoubleNode<T> tmp = find(element);

**if** (tmp == **null**)

{

**throw** **new** NoSuchElementException("DoubleList");

}

**else** **if** (count ==1)

{

head = tail = **null**;

}

**else**

{

DoubleNode<T> previous, next;

**if** (tmp == head) // if element = the first element

{

next = head.getNext();

next.setPrevious(**null**);

head = next;

}

**else** **if** (tmp == tail) // if element = the last element

{

previous = tail.getPrevious();

previous.setNext(**null**);

tail = previous;

}

**else**

{

previous = tmp.getPrevious();

next = tmp.getNext();

previous.setNext(next);

next.setPrevious(previous);

}

}

count--;

modCount++;

**return** tmp.getElement();

}

/\*\*

\*

\* **@return** The first element

\*/

**public** T first()

{

**return** head.getElement();

}

/\*\*

\*

\* **@return** The last element

\*/

**public** T last()

{

**return** tail.getElement();

}

/\*\*

\*

\* **@return** True if empty

\*/

**public** **boolean** isEmpty()

{

**return** (count == 0);

}

/\*\*

\*

\* **@return** The size of teh collection

\*/

**public** **int** size()

{

**return** count;

}

/\*\*

\* Creates a iterator for the collection

\*

\* **@return** the iterator created

\*/

**public** Iterator<T> iterator()

{

**return** **new** DoubleIterator();

}

/\*\*

\* Find a certain element

\*

\* **@param** target Element to be found

\* **@return** The node which contains the element

\*/

**public** DoubleNode<T> find(T target)

{

DoubleNode<T> result = **null**;

DoubleNode<T> current = head;

**if** (!isEmpty())

{

**while** (result == **null** && current != **null**)

{

**if** (current.getElement().equals(target))

{

result = current;

}

**else**

{

current = current.getNext();

}

}

}

**return** result;

}

/\*\*

\* Checks to see if a certain element is in the collection

\*

\* **@param** target The target element

\* **@return** True if found

\*/

**public** **boolean** contains(T target)

{

**return** (find(target) != **null**);

}

/\*\*

\* Converts the last to Strings

\* **@return** The String which contains all the elements in the collection

\*/

**public** String toString()

{

String result = "";

**if** (count == 0)

{

result = "Empty";

}

**else** **if** (count == 1)

{

result = (String)head.getElement();

}

**else**

{

DoubleNode<T> tmp = head.getNext();

result = (String)head.getElement();

**while** (tmp.getNext() != **null**)

{

result = result + "\n" + tmp.getElement();

tmp = tmp.getNext();

}

result = result + "\n" + tmp.getElement();

}

**return** result;

}

/\*\*

\* A private class that contains the iterator for this collection

\* **@author** Matt

\*

\*/

**private** **class** DoubleIterator **implements** Iterator<T>

{

**private** **int** iteratorModCount;

**private** DoubleNode<T> current;

/\*\*

\* Create the iterator

\*/

**public** DoubleIterator()

{

current = head;

iteratorModCount = modCount;

}

/\*\*

\* Checks to see if the iterator has another item.

\*

\* **@return** True if there is another item

\*/

**public** **boolean** hasNext() **throws** ConcurrentModificationException

{

**if** (iteratorModCount != modCount)

{

**throw** **new** ConcurrentModificationException();

}

**return** (current != **null**);

}

/\*\*

\* returns the next item in the iteration

\*/

**public** T next() **throws** NoSuchElementException

{

**if** (hasNext() == **false**)

{

**throw** **new** NoSuchElementException();

}

T result = current.getElement();

current = current.getNext();

**return** result;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DoubleNode.java Matt Matuk

// CSIT 211 Project 15.15 pg 616

// 1. This class is a double node that will contains a reference

// to the next and previous nodes.

// 2. This class will also contain a element

// 3. All methods and variables are generic

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **class** DoubleNode<T>

{

**private** DoubleNode<T> next, previous;

**private** T element;

/\*\*

\* Creates a Double linked node with a element

\*/

**public** DoubleNode()

{

element = **null**;

next = previous = **null**;

}

/\*\*

\* Creates a double linked node with a passed element

\*

\* **@param** elem The element to be set for this node

\*/

**public** DoubleNode(T elem)

{

element = elem;

next = previous = **null**;

}

/\*\*

\* Sets the next node

\*

\* **@param** node Node to be set

\*/

**public** **void** setNext(DoubleNode<T> node)

{

next = node;

}

/\*\*

\* Sets the previous node

\*

\* **@param** node The node to be set

\*/

**public** **void** setPrevious(DoubleNode<T> node)

{

previous = node;

}

/\*\*

\* Gets the next node

\*

\* **@return** The next node

\*/

**public** DoubleNode<T> getNext()

{

**return** next;

}

/\*\*

\* Gets the previous node

\*

\* **@return** The previous node

\*/

**public** DoubleNode<T> getPrevious()

{

**return** previous;

}

/\*\*

\* Gets the element for this node

\*

\* **@return** The element

\*/

**public** T getElement()

{

**return** element;

}

/\*\*

\* Sets the element for this node

\*

\* **@param** elem Element to be set

\*/

**public** **void** setElement(T elem)

{

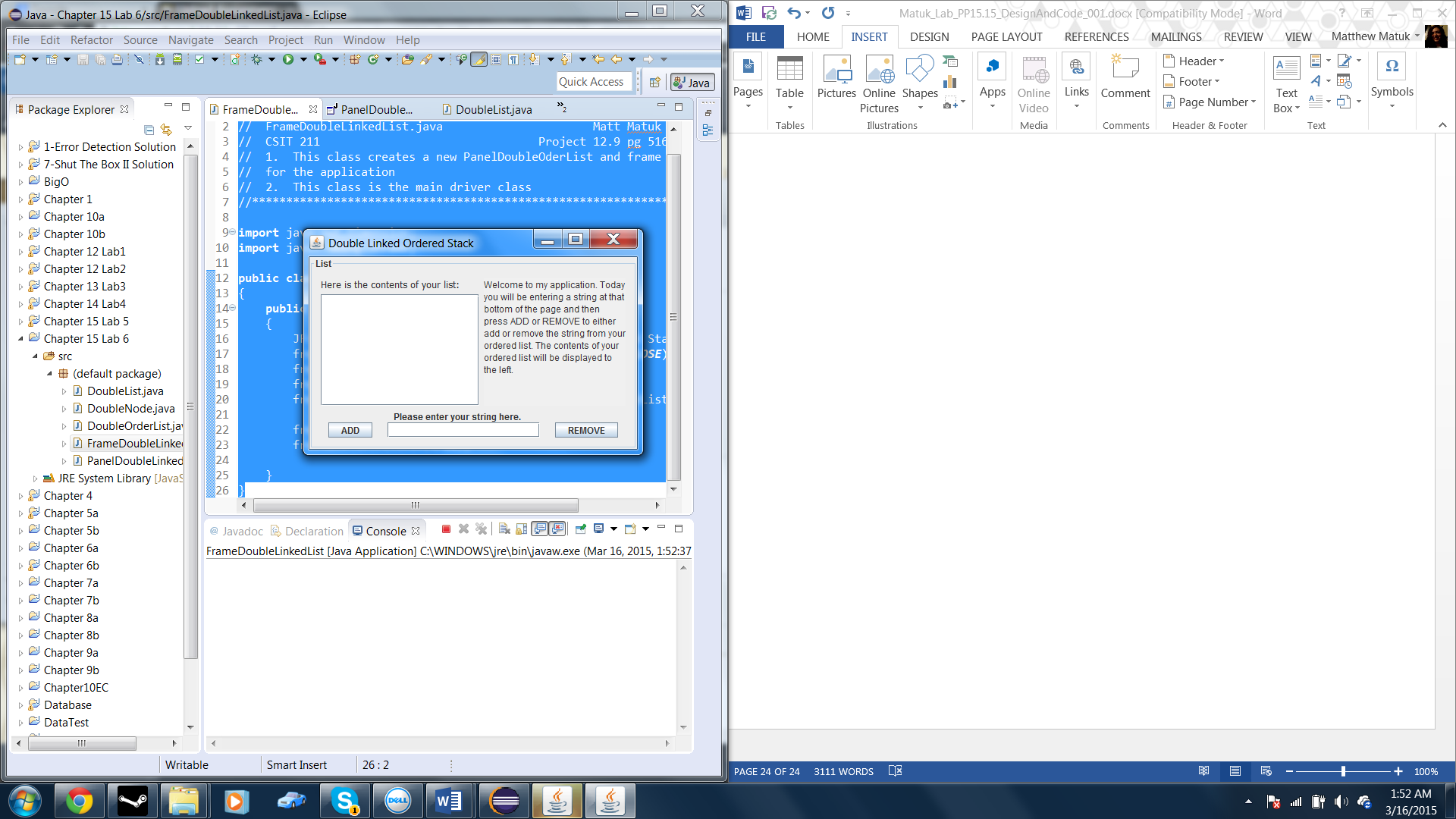
element = elem;

}

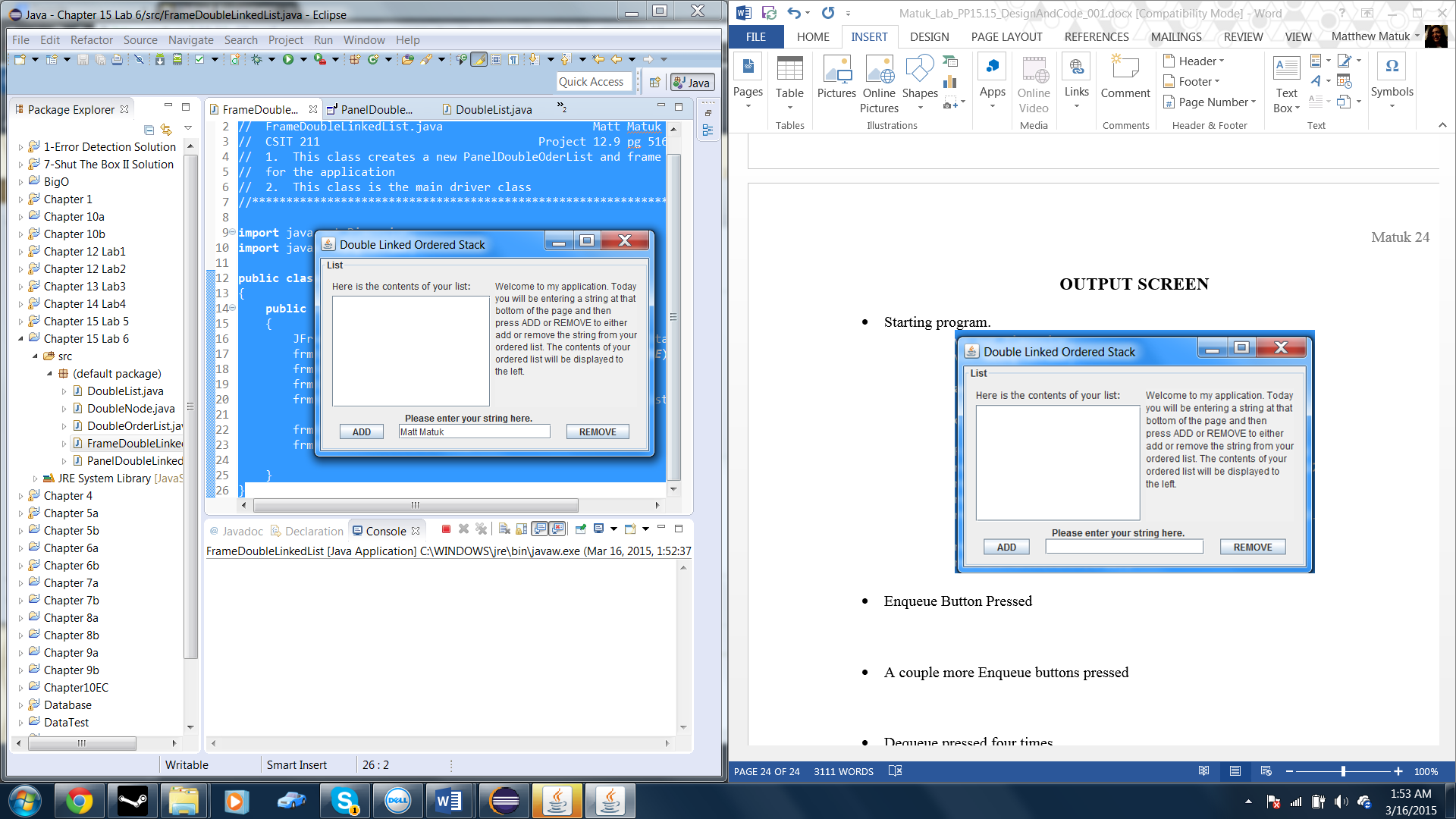
}

**OUTPUT SCREEN**

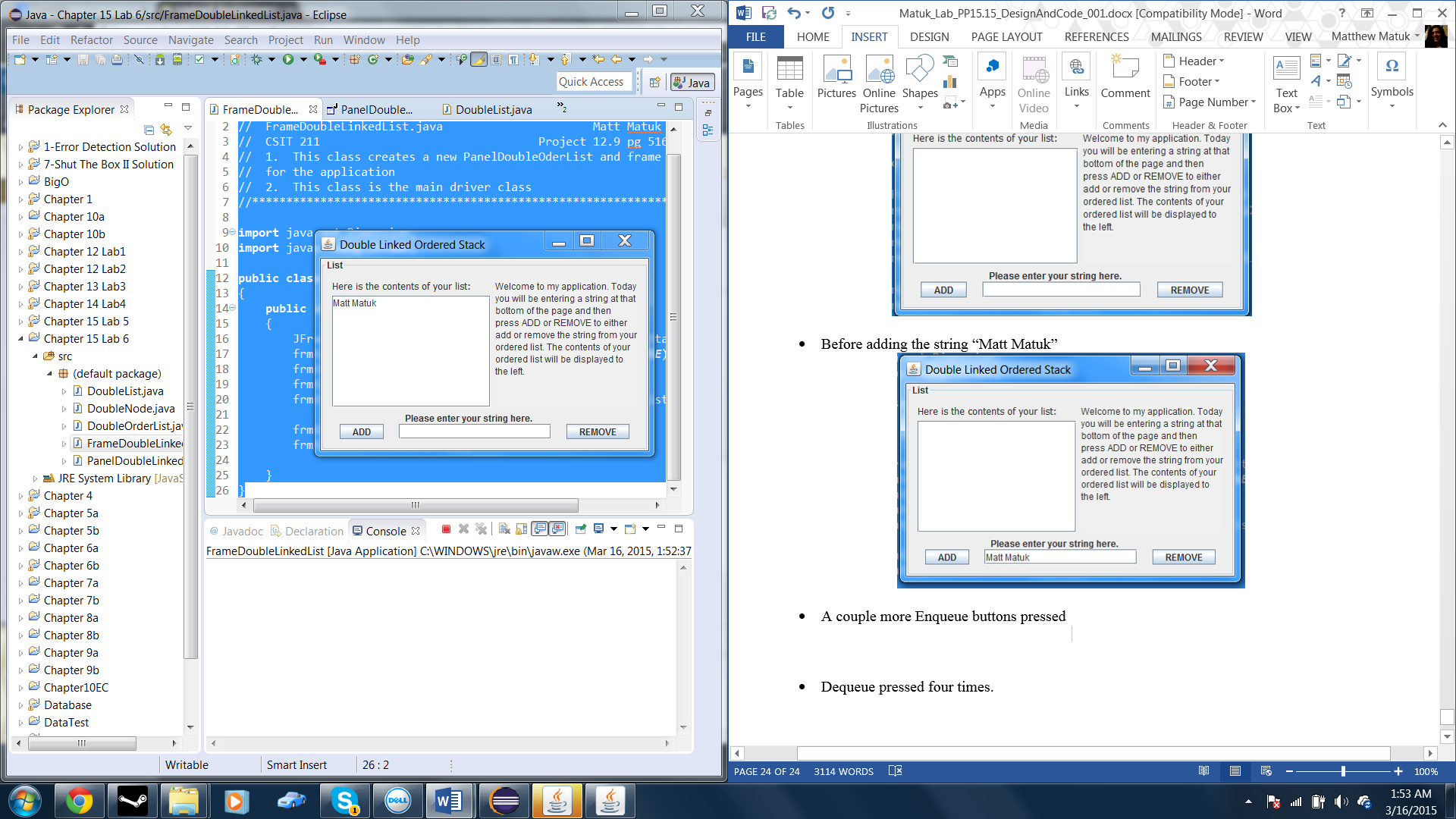
* Starting program.



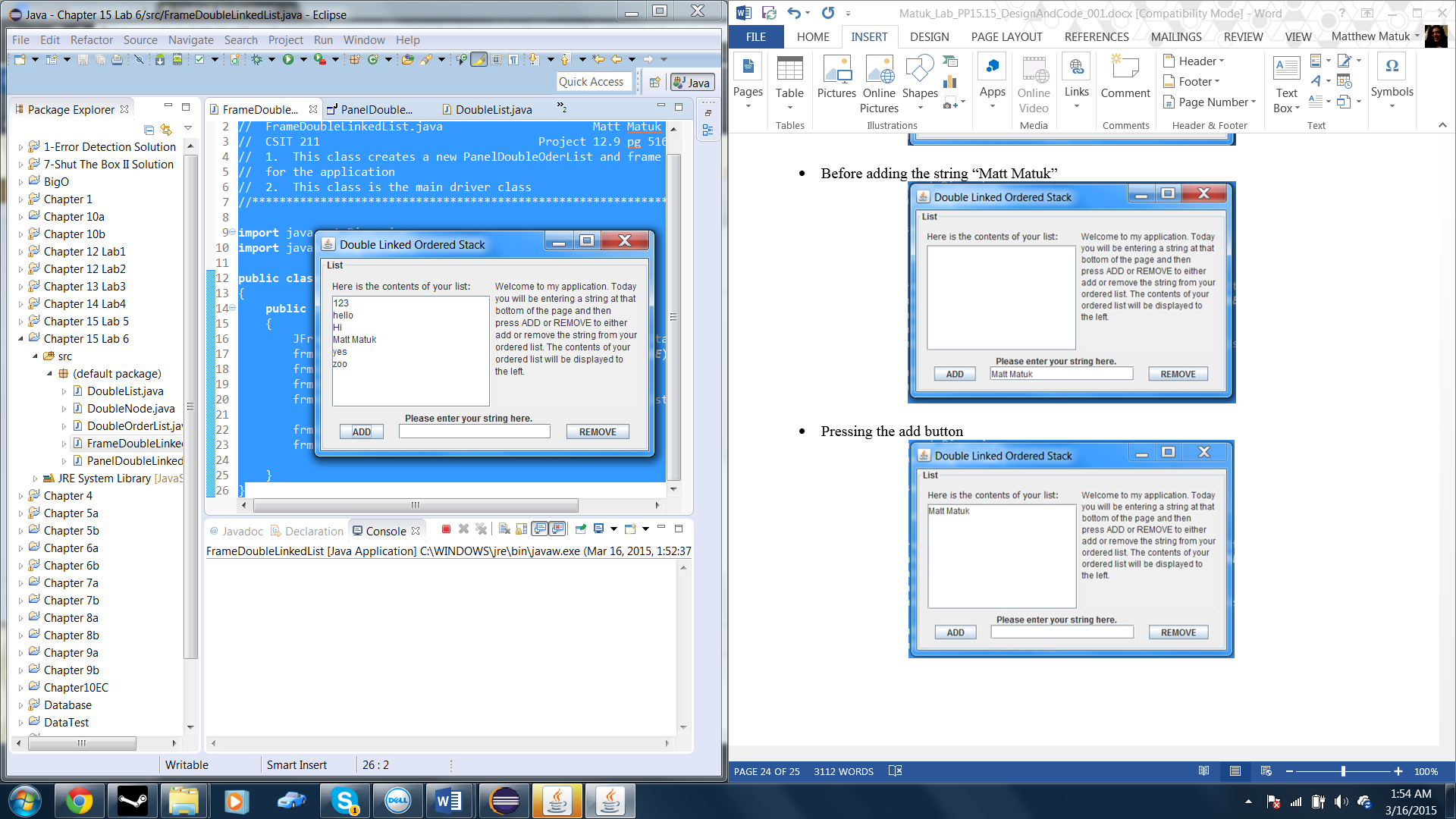
* Before adding the string “Matt Matuk”



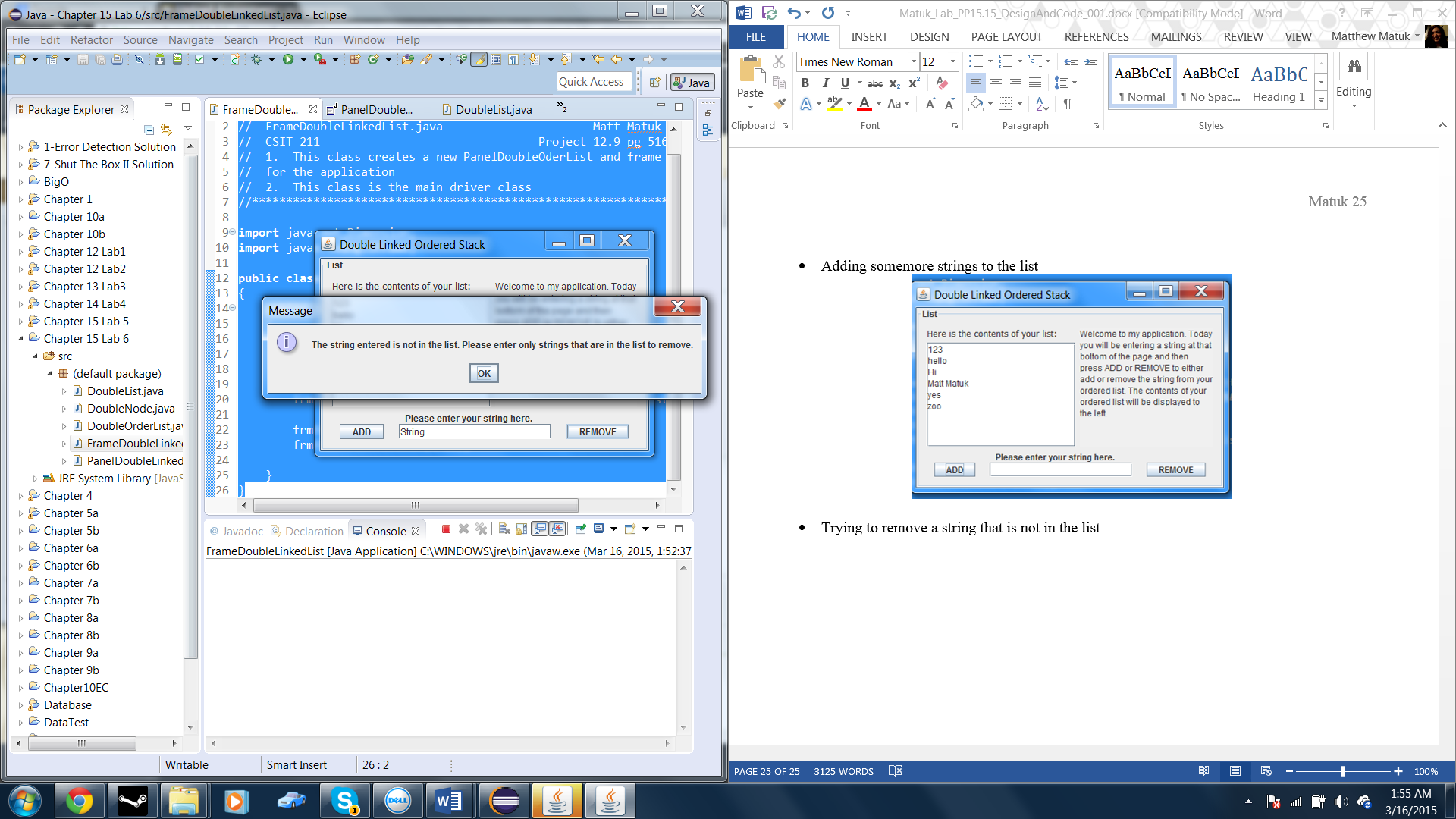
* Pressing the add button



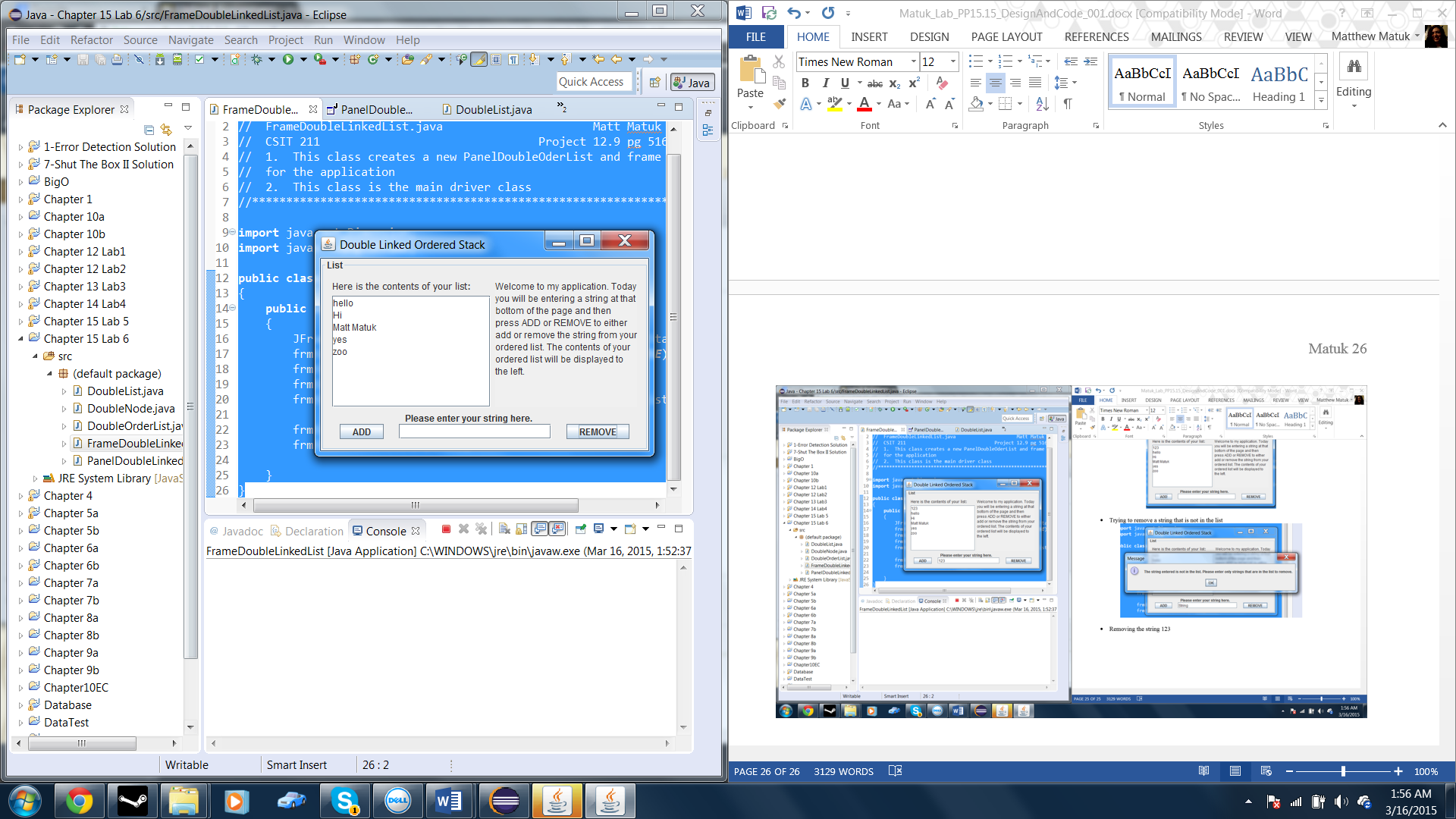
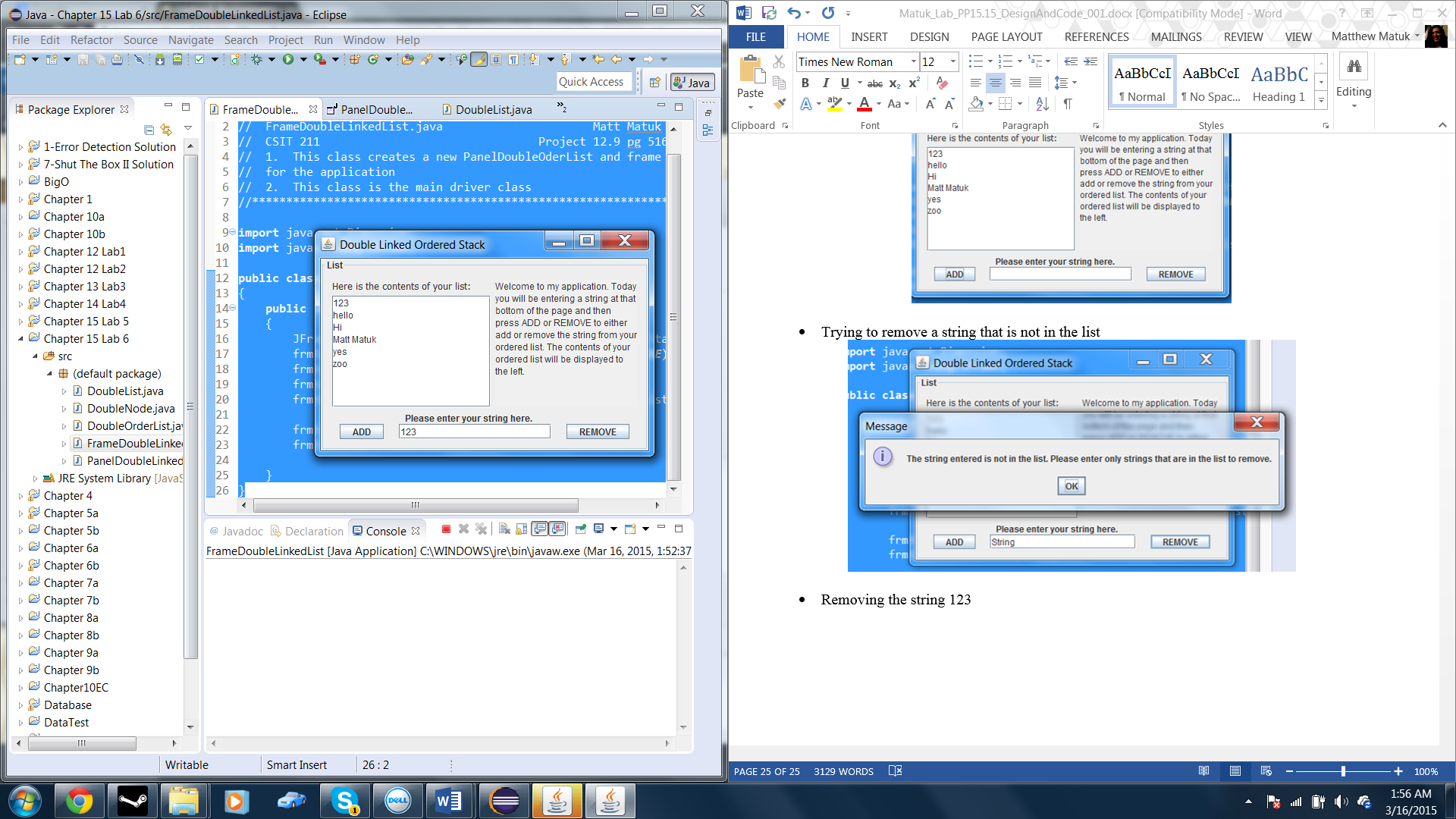
* Adding somemore strings to the list



* Trying to remove a string that is not in the list



* Removing the string 123



* Removing all the strings

